UNIVERSITÉ DE MONTRÉAL

CENTRE DE RECHERCHE SUR LES TRANSPORTS

CASE POSTALE 6128, SUCCURSALE A, MONTRÉAL, QUÉBEC, H3C 3J7

ÉTUDE DE L'IMPLANTATION ET DE LA COMPLEXITÉ DE CERTAINS PROBLÈMES DE PROGRAMMATION DYNAMIQUE DISCRÈTE

par

Martin Desrochers et François Soumis

Cette étude a été publiée grâce à une subvention du fonds F.C.A.R. pour l'aide et le soutien à la recherche.

Université de Montréal
Centre de recherche sur les transports - Publication #451F
Mars 1986

Dépôt légal - 2e trimestre 1986

FORMULAIRE DE DOCUMENTATION POUR RAPPORT

1. Titre

Étude de l'implantation et de la complexité de certains problèmes de programmation dynamique discrète
(Computational and Complexity Aspects of Discrete Dynamic Programming)

2. Date du rapport
Mars 1986

3. No de rapport du C.R.T.
451 F

4. No I.S.B.N.

5. Auteur(s)
Martin Desrochers et François Soumis

6. Nom et adresse de l'organisme subventionnaire

7. Résumé Dans les problèmes de programmation dynamique, nous devons résoudre des équations de récurrences pour tous les états. La solution optimale est un ensemble de chemins entre états initiaux et états finaux. Nous étudions une classe de problèmes de programmation dynamique discrète : les problèmes de plus court chemin avec plusieurs contraintes supplémentaires. Nous ferons ici la comparaison entre deux classes d'algorithmes résolvant ces problèmes : les algorithmes de reaching et les algorithmes de pulling. Nous décrirons une implantation de l'algorithme de reaching et deux implantations de l'algorithme de pulling. Nous comparerons leur complexité et présenterons les résultats de tests numériques sur des problèmes de grande taille. La principale conclusion de ces tests est que le temps d'exécution d'une des implantations du pulling ne croît pas avec le nombre d'états mais plutôt diminue lorsque le nombre d'états associés à des chemins réalisables diminue lors de l'augmentation du nombre de contraintes supplémentaires.

8. Mots clés :
programmation dynamique, complexité, plus courts chemins avec contraintes.

9. Nombre de pages : 24

10. Code de prix : A01

SOMMAIRE

Dans les problèmes de programmation dynamique, nous devons résoudre des équations de récurrences pour tous les états. La solution optimale est un ensemble de chemins entre états initiaux et états finaux. Nous étudions une classe de problèmes de programmation dynamique discrète : les problèmes de plus court chemin avec plusieurs contraintes supplémentaires. Nous ferons ici la comparaison entre deux classes d'algorithmes résolvant ces problèmes : les algorithmes de reaching et les algorithmes de pulling. Nous décrirons une implantation de l'algorithme de reaching et deux implantations de l'algorithme de pulling. Nous comparerons leur complexité et préenterons les résultats de tests numériques sur des problèmes de grande taille. La principale conclusion de ces tests est que le temps d'exécution d'une des implantations du pulling ne croît pas avec le nombre d'états mais plutôt diminue lorsque le nombre d'états associés à des chemins réalisables diminue lors de l'augmentation du nombre de contraintes supplémentaires.

INTRODUCTION

Tous les problèmes de programmation dynamique discrète sont équivalents à des problèmes de plus court chemin. Chaque état est équivalent à un noeud et chaque transition entre deux états définit un arc entre deux noeuds. Nous considérons dans cet article le cas spécial des problèmes de plus court chemin avec contraintes supplémentaires (ou multi-critère). Dans ces problèmes, plusieurs états sont associés à chaque noeud et un arc entre deux noeuds indique une transition possible entre les états associés à chacun des noeuds.

1. LE PROBLÈME

Soit le réseau G=(N,A) ou A est l'ensemble des arcs et N est l'ensemble de noeuds dont une source p et un puits q. Le coût $c_{ij} \in R$ positif ou négatif, associé à chaque arc $(i,j) \in A$ permet de définir le problème de plus court chemin (1-3).

$$\text{Minimiser} \quad c.x = \sum_{(i,j)\in A} c_{ij}\, x_{ij} \tag{1}$$

sujet à

$$x \in X(G) = \begin{cases} \sum_{j\in N} x_{ij} - \sum_{j\in N} x_{ji} = \begin{cases} +1, & i=p \\ 0, & i\in N,\ i\neq p,\ i\neq q \\ -1, & i=q \end{cases} & (2) \\ x \geq 0 & (3) \end{cases}$$

Dans notre problème, nous avons des contraintes sur l'utilisation de L ressources. À chaque arc (i,j) et à chaque ressource ℓ, est associée $d^{\ell}_{ij} \in N$, la quantité de ressource ℓ utilisée par l'arc (i,j). Lorsque les noeuds sont des tâches, le coût et les ressources utilisées par un noeud i sont ajoutés à ceux des arcs extrants (i,j). Soit T^{ℓ}_{j}, la quantité de ressource ℓ utilisée en poursuivant un chemin depuis la source jusqu'au noeud j; notre problème consiste à trouver le chemin élémentaire de coût minimum

entre p et q tel que T_j^ℓ respecte la fenêtre de réalisabilité $[a_j^\ell, b_j^\ell]$ pour tous les noeuds du chemin et pour toutes les ressources $\ell = 1,\ldots,L$. Nous considérons le problème où il est possible d'atteindre un noeud en utilisant moins d'une ressource ℓ que le minimum de la fenêtre de réalisabilité de cette ressource. En gaspillant la ressource ℓ, il est alors possible de débuter la visite de ce noeud au ninimum de la fenêtre de réalisatilité de cette ressource. Le problème du plus court chemin avec fenêtre de temps [DESR 83] en est un exemple; il est possible d'arriver à un noeud avant le début de la fenêtre de temps et d'attendre, i.e. gaspiller du temps, avant de débuter la visite du noeud au début de la fenêtre de temps.

Pour obtenir le problème de plus court chemin avec contraintes supplémentaires (PPCCCS), nous devons ajouter au problème de plus court chemin (1-3) les contraintes suivantes :

$$x_{ij} > 0 \Rightarrow T_i^\ell + d_{ij}^\ell \leq T_j^\ell \qquad (i,j,)\in A,\ \ell=1,..,L \qquad (4)$$

$$a_i^\ell \leq T_i^\ell \leq b_i^\ell \qquad i\in N,\ \ell=1,\ldots,L \qquad (5)$$

où x_{ij} représente le flot sur l'arc (i,j) et T_i^ℓ, la quantité de la $\ell^{ième}$ ressource utilisée pour parvenir au noeud i.

Si le gaspillage de ressource n'est pas permis, il faut remplacer l'inégalité de l'équation (4) par une égalité. Le problème résultant est plus contraint et sa solution optimale peut posséder un coût plus élevé. Ce modèle correspond à la programmation dynamique classique (voir Jaffe [JAFF84]) et est généralement plus complexe à résoudre que si le gaspillage est permis.

Nous considérons que tous les arcs non-réalisables sont exclus de l'ensemble A. Un arc (i,j) est non-réalisable s'il ne permet pas de passer du noeud i au noeud j en respectant les fenêtres de réalisabilité en i et j. Tous les arcs $(i,j)\in A$ respectent donc la condition suivante pour toutes les ressources :

$$a_i^\ell + d_{ij}^\ell \leq b_j^\ell \qquad \ell=1,\ldots,L \qquad (6)$$

Les contraintes supplémentaires (4-5) peuvent aussi prendre la forme :

$$\sum_{(i,j)\in A} d_{ij}^{\ell} x_{ij} \leq b_q^{\ell} \qquad \ell=1,\ldots,L \qquad (7)$$

Aneja et al. [ANEJ83], pour résoudre des problèmes exprimés sous la forme (1-3)-(7), les transforment en problèmes équivalents exprimés sous la forme (1-5). Toutefois, la transformation inverse n'est pas possible, la forme (1-5) comportant des contraintes à chaque noeud au contraire de la forme (1-3,7) qui ne comporte que des contraintes sur le dernier noeud du chemin (puits).

2. REVUE DE LA LITTÉRATURE

Bien que le problème du plus court chemin avec contraintes supplémentaires et le problème voisin du plus court chemin multicritère, aient fait l'objet de près d'une quinzaine d'articles, nous ne discuterons ici que de ceux utilisant notre approche : la programmation dynamique. Pour une revue de littérature plus complète, voir Desrochers [DESR86].

Introduisons d'abord une définition des états commune à la plupart des approches par programmation dynamique pour ces problèmes. À chaque chemin X_{pj} de l'origine à un noeud j, est associée une étiquette (quantité de ressources utilisées, coût) correspondant aux L quantités de ressources utilisées T_j^{ℓ} et au coût du chemin X_{pj}. Ces étiquettes sont représentées par $(T_j^{1k}, T_j^{2k}, \ldots, T_j^{Lk}, C_j^k)$ et dénotent les caractéristiques du $k^{ième}$ chemin X_{pj}^k, les indices k pourront être absents si le contexte n'est pas ambigu. De plus, $T_j^{\cdot k}$ ou $T_j^{\cdot}$ seront utilisés pour représenter le vecteur de dimension L. Pour le traitement de ces problèmes par programmation dynamique, l'ensemble des états est $\{T_j^{\cdot k}$ pour tous k et j$\}$ et C_j^k est le coût associé à l'état $T_j^{\cdot k}$. Un chemin X_{pj}^k et l'état associé $T_j^{\cdot k}$ sont réalisables si $a_j^{\ell} \leq T_j^{\ell} \leq b_j^{\ell}$ pour toutes les ressources $\ell=1,\ldots,L$. Dans le cas des problèmes de

plus court chemin multicritère, la définition des états est la même mais l'algorithme optimise par rapport à toutes les composantes de l'état.

Calcul des successeurs (reaching)

Il existe deux méthodes de ce type. La première méthode, basée sur la généralisation de l'algorithme de Ford-Bellman [JOKS66, VINC74, HABE81, DESR 83, JAFF84], fonctionne par correction des étiquettes. La seconde méthode, basée sur la généralisation de l'algorithme de Dijkstra [MINO75, HANS80, ANEJ83, DESR86], fonctionne par marquage permanent des étiquettes. L'étape de base dans les deux algorithmes, le reaching, consiste, à partir d'une étiquette associée à un chemin X_{pi}, à obtenir les étiquettes correspondant aux chemins réalisables X_{pj} prolongeant le chemin X_{pi} par tous les arcs $(i,j) \in A$. Ces algorithmes crêent pour chaque noeud i de nouvelles étiquettes à plusieurs noeuds j; ainsi des étiquettes sont créées à plusieurs reprises à chaque noeud j si le noeud j possède plusieurs prédécesseurs. Il faut donc effectuer plusieurs mises à jour des étiquettes à chaque noeud j. Cette mise à jour est coûteuse surtout s'il y a plusieurs contraintes supplémentaires.

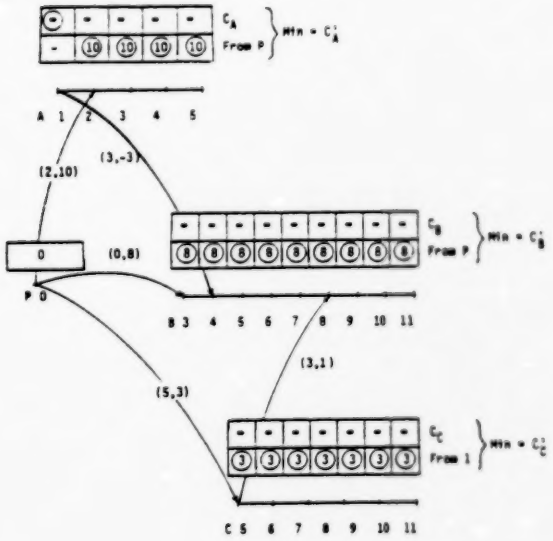

Figure 1
Reaching à partir du noeud P

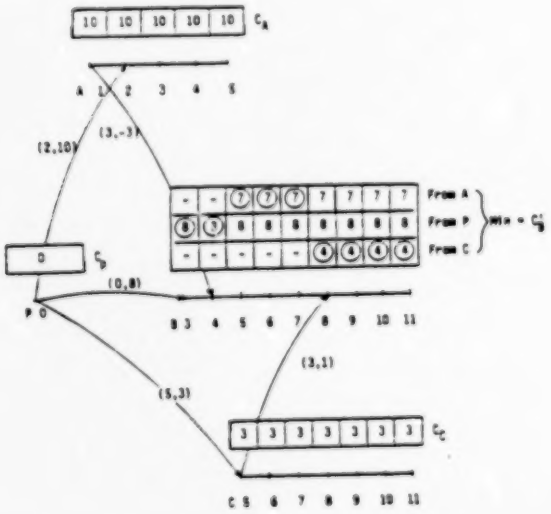

Figure 2
Pulling vers le noeud B

La figure 1 illustre une étape de l'algorithme de reaching pour le cas d'une contrainte supplémentaire. À chaque noeud, nous connaissons la valeur actuelle de C_j pour chaque état réalisable. Nous effectuons le reaching pour l'état (0,0) du noeud P. Aux noeuds A, B et C, le coût C'_j de chaque état est indiqué par un coût encerclé et est le minimum du coût actuel C_j et de $C_p + c_{pj}$. Il y a modification des coûts aux trois noeuds.

Recherche des prédécesseurs (pulling)

Cette méthode [DESR86] primale-duale, mise au point pour la réoptimisation du problème du plus court chemin avec fenêtre de temps, calcule les chemins X_{pj} associés à une partie ou à tous les états d'un noeud j. Pour un ensemble S d'états du noeud j, le pulling consiste à créer les étiquettes $(T_j^* = (\max\{a_j^\ell, T_j^{\ell k}+d_{ij}^\ell\}, \ell=1,\ldots,L), C_i^k+c_{ij})$ associées aux chemins X_{pj} obtenus en prolongeant tous les chemins X_{pi} auxquels l'ajout de l'arc $(i,j)\in A$ permet l'arrivée au noeud j en un état $T_j^*\in S$. Les nouvelles étiquettes étant créées à un seul noeud, cet algorithme demande, au contraire des algorithmes basés sur le reaching, la mise à jour des étiquettes à un seul noeud.

La figure 2 représente une étape de l'algorithme de pulling. Nous effectuons le pulling au noeud B pour calculer les nouveaux coûts C'_B des états T_B. Le coût C'_B est indiqué par un coût encerclé et est le minimum des coûts obtenus par prolongement des chemins depuis les noeuds A, C et P.

Dominance et principe d'optimalité

Avant de débuter la définition des deux algorithmes, nous devons définir deux notions : la dominance et les équations de récurrence de PCCCS.

Définition 1

Soit X^1 et X^2 deux chemins distincts de p à j et leurs étiquettes associées (T_j^{*1}, C_j^1) et (T_j^{*2}, C_j^2), X^1 domine X^2 ou $(T_j^1, C_j^1) < (T_j^2, C_j^2)$ si et

seulement si $C_j^1 \leq C_j^2$, $T_j^{\ell 1} \leq T_j^{\ell 2}$ pour tout $\ell=1,\ldots,L$ et $(T_j^{*1}, C_j^1) \neq (T_j^{*2}, C_j^2)$.

La relation de dominance nous permet de conclure qu'un chemin X_{pj} et son étiquette associée (T_j^*, C_j) sont non dominés (efficaces) si et seulement si X_{pj} est le chemin de coût minimal arrivant au noeud j avec des valeurs inférieures ou égales à T_j^ℓ $(\ell=1,\ldots,L)$.

Nous utilisons le principe d'optimalité de Bellman, i.e. que tout sous-chemin d'un chemin efficace doit lui aussi être efficace, pour définir les équations de récurrence suivantes qui caractérisent les chemins efficaces de la source à tous les noeuds pour le PPCCC :

Pour tout $j \in N$ et T_j^* réalisable :

$$C(j,T_j^*) = \min_{\substack{i\in N,\ T_j^* \text{ état réalisable}}} \{C(i,T_i + c_{ij}) \mid T_i^\ell + d_{ij}^\ell \leq T_j^\ell, (i,j)\in A,\ \ell=1,\ldots,L\} \qquad (8)$$

$$C(p,0) = 0 \qquad (9)$$

L'application du principe d'optimalité permet d'accélérer le calcul des équations de récurrence (8-9) en évitant de prolonger des chemins X_{pi}^k dominés et de produire ainsi de nouveaux chemins X_{pj} dominés.

3. UN ALGORITHME DE REACHING MULTIDIMENSIONNEL

Étape 1 : Initialisation

$$Q_i = \begin{cases} \{(0..0)\} & i=p \\ \emptyset & \text{pour tout } i\in N,\ i\neq p \end{cases}$$

$$P_i = \emptyset \qquad \text{pour tout } i\in N$$

Étape 2 : Choix de l'étiquette à traiter

Trouver l'étiquette $((T_i^{\ell k}, \ell=1,\ldots,L), C_i^k)$ de coût lexicographique minimum parmi l'ensemble des étiquettes temporaires $T = \bigcup_{i\in N} (Q_i - P_i)$. Si $T=\emptyset$, terminer.

Étape 3 : Traitement de l'étiquette $((T_i^{\ell k}, \ell=1,\ldots,L), C_i^k)$

Pour tous les successeurs j du noeud i faire

début

si $T_i^{\ell k} + d_{ij}^{\ell} \leq b_j^{\ell}$ (respect des fenêtres de réalisabilité)

alors $(T_j^*, C_j) = ((\max(a_j^{\ell}, T_i^{\ell k} + d_{ij}^{\ell}), \ell=1,\ldots,L), C_i^k + c_{ij})$

$Q_j = \text{EFF}(Q_j \cup (T_j^*, C_j))$

fin.

$P_i = P_i \cup \{(T_i^{*k}, C_i^k)\}$

aller à l'étape 2.

De façon analogue à l'algorithme de Dijkstra, cet algorithme termine si tous les arcs (i,j) ont une longueur lexicographique positive.

La complexité de l'algorithme présenté est dépendante du nombre d'états réalisables. Soient

L : le nombre de contraintes supplémentaires,

$n = |N|$: le nombre de noeuds dans le graphe G,

$v = \max_{i \in N} \{\prod_{\ell=1}^{L} (b_i^{\ell} - a_i^{\ell} + 1)\}$: le nombre maximum d'états à un noeud,

$V = \sum_{i=1}^{L} \prod_{\ell=1}^{L} (b_i^{\ell} - a_i^{\ell} + 1)$: le nombre total d'états.

L'implantation réalisée utilise une liste pointée triée en ordre lexicographique pour représenter Q_i. L'espace mémoire utilisé est en pire cas $O(|A|+V)$. Toutefois, les listes ne sont efficaces que si une faible proportion des états existe, permettant de réduire la mémoire nécessaire et le travail à effectuer.

L'étape 1 est exécutée une fois et est d'ordre n. L'étape 2 est exécutée au maximum V fois, chaque exécution demande au maximum O(n) opérations. Le traitement de l'étiquette choisie (étape 3) est effectuée au maximum V fois et le nombre de successeurs pour l'étiquette traitée est au

maximum de n. Pour chacun de ces successeurs, la fusion et la réduction à un ensemble d'étiquettes non-dominées associées au noeud i demande la comparaison de la nouvelle étiquette avec au maximum les $b_i - a_i + 1$ étiquettes possiblement présentes. Globalement, pour tous les successeurs, cette opération demande O(VL) opérations. La complexité de cette implantation est d'ordre $O(V^2 L)$. Toutefois, le travail à effectuer est proportionnel au carré du nombre d'étiquettes réellement présentes dans la solution. Cette implantation est intéressante si ce nombre est faible par rapport à V.

4. L'ALGORITHME DE PULLING MULTIDIMENSIONNEL

L'algorithme de pulling est un algorithme primal-dual. L'ensemble P_j contient des étiquettes primales, i.e. réalisables et qui sont des bornes supérieures sur le coût des états. L'ensemble Q_j contient des étiquettes dualies, i.e. des bornes inférieures sur le coût des états. Le but de cet algorithme est de toujours réduire l'écart entre la borne supérieure et la borne inférieure. Pour réduire cet écart, l'algorithme effectue un pulling primal pour calculer la nouvelle borne supérieure et un pulling dual pour calculer la nouvelle borne inférieure. En fait, ces deux étapes sont effectuées simultanément et Desrochers [DESR86] démontre que si tous les arcs ont une longueur lexicographique positive le nombre d'états avec un écart entre les bornes inférieure et supérieure diminue à chaque pulling.

Étape 1. Initialisation

- préparation des ensembles P_j et Q_j.
- calcul des $(d^*_{.j}, c_{.j})$ pour chaque noeud.

Étape 2. Choix d'une étiquette (T_j^{*k}, C_j^k) à traiter

Trouver l'étiquette (T_j^{*k}, C_j^k) de coût lexicographique minimum parmi l'ensemble $S = \bigcup_{j \in N} (Q_j - P_j)$. Si $S = \emptyset$, terminer.

Étape 3. Choix de la zone à explorer parmi la zone d'incertitude

La fin de la zone à explorer TSUP est définie par

$$TR_j = \text{minlex}\ \{b^{\bullet}_j\} \cup \{T^{\bullet m}_j \mid (T^{\bullet m}_j, C^m_j) \in P_j \text{ et } T^{\bullet m}_j \underset{>}{L} T^{\bullet k}_j\}$$

$$TSUP = \text{minlex}\ \{T^{\bullet k}_j + d^{\bullet}_{\bullet j}, TR_j\}.$$

Étape 4. Remplacement de l'étiquette $(T^{\bullet k}_j, C^k_j)$

A) Calcul de la nouvelle fonction duale dans la zone à explorer

$$R_j = EFF\{(T^{\bullet}_j, C_j) = ((\max\{a^{\ell}_j, T^{\ell k}_i + d^{\ell}_{ij}\}, \ell = 1, \ldots, L), C^k_i + c_{ij})$$

pour tout $(i,j) \in A$, $(T^{\bullet k}_i, C^k_i) \in Q_i$

et $T^{\bullet}_j$ est dans la zone à explorer.$\}$

B) Calcul de la fonction primale dans la zone à explorer

$$RP_j = EFF\{(T^{\bullet}_j, C_j) = ((\max\{a^{\ell}_j, T^{\ell k}_i + d^{\ell}_{ij}\}, \ell = 1, \ldots, L), C^k_i + c_{ij})$$

pour tout $(i,j) \in A$, $(T^{\bullet k}_i, C^k_i) \in P_i$

et $T^{\bullet}_j$ est dans la zone à explorer.$\}$

C) Création des nouveaux ensembles Q_j et P_j

- $P_j = P_j \cup (R_j \cap RP_j)$
- $Q_j = (Q_j - \{(T^{\bullet k}_j, C^k_j)\}) \cup (R_j)$

retourner à l'étape 3.

Nous utiliserons pour exprimer la complexité de l'algorithme de pulling les notation définies à la section précédente. Nous étudierons la complexité de deux implantations de l'algorithme de pulling. Ces implantations ne conservent en mémoire centrale que les étiquettes efficaces dans un arbre cousu auto-ajustant (SPLAY) mis au point par Sleator et Tarjan [SLEA83] permettant d'accéder à un élément et à son successeur. Sleator et Tarjan montrent que la complexité de toute séquence de m opérations sur un arbre de v étiquettes (avec $m > v$) est d'ordre $m \log v$. Ce résultat permet,

en moyenne, d'accéder à une étiquette en temps logarithmique. Selon les résultats obtenus précédemment, seulement 5% à 15% des étiquettes sont efficaces et l'utilisation de cet arbre permet une économie appréciable de mémoire.

La complexité de l'algorithme de pulling dépend de l'implantation utilisée. Toutefois la complexité des trois premières étapes de l'algorithme est indépendante de la structure de données choisie.

L'initialisation, lors de l'étape 1, prépare les ensembles P_i et Q_i. La fabrication d'un tas (heap) comprenant les éléments lexicographiquement minimum de chacun des ensembles Q_i demande O(Lnlogn) opérations. Lors de l'étape 1, il est aussi nécessaire de déterminer, par un examen de tous les arcs, l'arc lexicographiquement minimum à chaque noeud, ce qui demande $O(|A|L)$ opérations. La complexité, en pire cas de l'étape 1, est globalement de $O(L.(|A|+n+n\log n)$ opérations.

L'utilisation d'un tas (heap) permettant l'accès rapide à l'élément lexicographiquement minimum des ensembles Q_i permet de simplifier les étapes 2 et 3. Dans le cas simple, le choix de l'étiquette à remplacer se limite à mettre à jour le tas et à accéder à son sommet.

Cette étape sera exécutée au maximum V fois. Chaque exécution demande O(L.logn) opérations pour la mise à jour du tas et le choix de l'étiquette à traiter. Globalement, l'étape 2 demande dans le pire cas O(V.L.logn) opérations.

La troisième étape peut être exécutée au maximum V fois et demande le calcul de la frontière de la zone à explorer en O(L) opérations. Globalement, l'étape 3 demande en pire cas O(VL) opérations. Globalement, ces trois étapes demandent $O(|A|L+nL+VL\cdot\log n)$ opérations.

La quatrième étape peut être implantée de deux façons, selon l'algorithme utilisé pour trouver les vecteurs efficaces.

L'utilisation d'un arbre cousu auto-ajustant [SLEA83] pour conserver les étiquettes efficaces permet, en moyenne, un temps d'accès logarithmique aux étiquettes. L'utilisation d'une table de dimension V pour représenter les étiquettes associées aux états dans la zone d'incertitude permet de simplifier la vérification de la dominance et de profiter d'une partie des avantages d'une table, tout en conservant l'économie d'espace des arbres. La construction des bornes inférieure et supérieure dans la zone d'incertitude demande le calcul du coût de chacun des chemins en provenance des prédécesseur du noeud et ayant leur étiquette associée dans la zone d'incertitude. L'accès logarithmique aux étiquettes des prédécesseurs demande en moyenne O(logv) opérations. Cette moyenne peut être utilisée puisque le nombre d'accès aux étiquettes est O(v.nombre de successeurs d'un noeud) ce qui est plus grand que O(v). La vérification de la dominance s'effectuant dans la table locale, elle se limite à vérifier si le coût du chemin est plus petit que le coût actuel de l'état associé au chemin. Lors de la fabrication des nouveaux ensembles Q_j et P_j, il y a vérification de la dominance entre les étiquettes de la table et reconstruction de l'arbre cousu auto-ajustant. Globalement, puisqu'il y a au total V étiquettes et puisque chacune de ces étiquettes est traitée au maximum une fois, l'étape 4 demande pour le remplacement des étiquettes Q_j des n noeuds O(VL(nlogv+v)) opérations.

La complexité globale de la première implantation est d'ordre O(VL.(nlogv+v)) pour le pulling et d'ordre $O(V^2 L)$ pour le reaching. Puisque O(nlogv+v) < O(V), la complexité de cette implantation du pulling est moindre que celle de l'implantation pour le reaching.

La seconde implantation possible utilise aussi un arbre cousu auto-ajustant pour conserver les étiquettes efficaces. Toutefois, le test de dominance n'est pas effectué dans une table mais plutôt en utilisant un algorithme de calcul du minimum d'un ensemble de vecteurs (i.e. d'étiquettes multidimensionnelles) [KUNG75, DESR86]. La complexité de cet algorithme est, en pire cas, d'ordre $O(v\log^2 v)$ à chaque noeud. Globalement, pour les n noeuds, le calcul de la dominance demande $O(VL\log^2 v)$ opérations et l'accès aux prédécesseurs $O(VL\log v)$ opérations. L'étape 5 demande, pour cette implantation, $O(VL(n\log v+\log^2 v))$ opérations.

La complexité globale de la seconde implantation est d'ordre $O(VL(n\log v+\log^2 v))$ pour le pulling et d'ordre $O(V^2L)$ pour le reaching. Encore une fois, la complexité de l'implantation du pulling est moindre que celle du reaching.

5. RÉSULTATS NUMÉRIQUES POUR DES PROBLÈMES AVEC UNE CONTRAINTE SUPPLÉMENTAIRE (L=1)

L'algorithme de pulling utilisant la première implantation (PULL1) et l'algorithme de reaching (REACH) ont été implantés en FORTRAN 77. Ces deux algorithmes ont été implantés avec des seaux généralisés [DESR86]. Dans le cas des algorithmes de pulling, nous utilisons une zone à explorer plus grande que celle définie à l'étape 3. Des tests ont montré qu'en doublant la dimension de la zone à explorer, le temps d'exécution diminue de 25%. PULL1 et REACH ont été testés sur un ordinateur CDC CYBER 173 pour comparer leur performances. La comparaison entre les deux algorithmes s'effectuera en variant trois paramètres :

1. Le nombre de noeuds n composant le réseau. Nous résoudrons des problèmes de 500, 1000 et 2500 noeuds.
2. Le nombre moyen d'arcs par noeud. Nous choisirons aléatoirement parmi les arcs réalisables selon (6) de façon à obtenir au total environ 10*n, 25*n ou 100*n arcs dans le réseau.

3. La largeur des fenêtres de temps. Les débuts de fenêtres a_i seront distribués entre 0 et 100 selon une loi uniforme et la fenêtre de chaque noeud prendra une largeur de 10, 25, 50 ou 100 unités de temps.

Six problèmes test différents ont été générés pour chaque combinaison réalisable des trois paramètres. Ces problèmes ont été produits en utilisant la même méthode que Desrosiers, Pelletier et Soumis [DESR83].

Les problèmes ont été produits en utilisant les paramètres suivants :

1. les tâches sont dispersées dans un carré de [0,70] x [0,70] selon une loi uniforme;
2. les durées des tâches d_i sont dispersées dans l'intervalle [5,15] selon une loi uniforme;
3. le temps inter-tâches t_{ij} est égal à la distance euclidienne entre les taches i et j additionnée de la durée de la tâche i;
4. le coût inter-tâches c_{ij} est égal à t_{ij} moins une grande constante (33 333). Cette constante représente la valeur marginale d'un noeud lors de la génération de colonnes. La constante choisie rend le coût de presque toutes les routes de trois noeuds ou plus négatif.

L'ordinateur utilisé ne permettait pas de conserver en mémoire centrale la totalité des noeuds, des arcs et des étiquettes. Les noeuds et les étiquettes sont conservés en mémoire centrale, alors que les arcs sont divisés en pages et que l'on ne conserve qu'une seule page en mémoire centrale. Lors d'un défaut de page, i.e. lorsque l'exécution de l'algorithme nécessite des arcs d'une autre page, cette nouvelle page est amenée en mémoire centrale. Ce processus peut se comparer à celui de la mémoire virtuelle sur d'autres ordinateurs. Le temps de gestion de la mémoire est comptabilisé séparément du temps d'exécution de l'algorithme.

TABLEAU 1 Statistiques d'exécution des deux algorithmes

|N| réseau	D	|A|	$\|E_s\|$	REACH # problèmes résolus	REACH |E| traitées	REACH Temps Gestion mémoire Moy (sec.)	REACH Temps Gestion mémoire Max (sec.)	REACH Exécution Moy (sec.)	REACH Exécution Max (sec.)	PULL1 # problèmes résolus	$\|E_r\|$	PULL1 Temps Gestion mémoire Moy (sec.)	PULL1 Temps Gestion mémoire Max (sec.)	PULL1 Exécution Moy (sec.)	PULL1 Exécution Max (sec.)
500	5500	5044	763	6	763	–	–	.50	.57	6	502	–	–	.69	.70
500	5500	12487	932	6	932	.37	.47	1.12	1.37	6	503	.12	.15	1.10	1.14
500	5500	24982	1045	6	1045	.78	1.13	2.21	2.75	6	517	.38	.68	1.81	1.84
500	5500	49984	1160	6	1160	1.59	1.97	4.51	5.45	6	539	.88	1.20	3.16	3.20
500	5500	4995	1231	6	1231	–	–	.70	.83	6	683	–	–	1.10	1.15
500	13000	12527	1675	6	1675	.65	.93	1.87	2.33	6	850	.38	.37	2.14	2.30
500	13000	24907	2065	6	2065	1.59	2.22	4.17	5.17	6	994	.95	1.04	3.91	4.07
500	13000	49988	2435	6	2435	2.88	3.57	9.33	10.82	6	1112	1.95	2.06	7.40	7.60
500	13000	5033	2167	6	2167	–	–	1.22	1.48	6	1130	–	–	2.15	2.24
500	25500	12501	3309	6	3309	1.56	1.94	3.56	4.42	6	1394	.79	.89	4.52	4.84
500	25500	24975	4132	6	4132	3.25	3.63	8.67	10.56	6	1660	1.75	1.83	8.87	9.34
500	25500	49953	4843	6	4843	6.47	7.19	19.65	22.81	6	1944	3.79	3.93	18.04	19.20
500	25500	4952	4243	6	4243	–	–	2.52	3.12	6	1894	–	–	4.19	4.38
500	50500	12477	6196	6	6196	2.33	3.06	7.89	9.35	6	2502	1.64	1.89	9.37	9.72
500	50500	25074	7961	6	7961	5.59	6.60	20.05	24.17	6	3056	2.99	3.48	19.22	20.71
500	50500	49913	10408	6	10408	10.42	13.23	53.71	60.16	6	3551	5.17	5.32	41.16	43.24
1000	11000	9993	1526	6	1526	–	–	1.01	1.14	6	1004	–	–	1.37	1.39
1000	11000	25015	1827	6	1827	.84	1.01	2.33	2.76	6	1023	.41	.70	2.30	2.34
1000	11000	50129	2061	6	2061	1.95	2.43	4.68	5.55	6	1046	1.07	1.37	3.78	3.83
1000	11000	99739	2393	6	2393	4.19	5.17	10.08	11.90	6	1093	2.43	2.58	6.68	6.74
1000	26000	10018	2434	6	2434	–	–	1.43	1.73	6	1362	–	–	2.20	2.33
1000	26000	25058	3575	6	3575	1.34	1.78	4.33	5.24	6	1745	.94	.96	4.56	4.76
1000	26000	50147	4278	6	4278	2.88	3.93	9.63	10.97	6	2022	2.17	2.35	8.22	8.33
1000	26000	100103	5077	6	5077	6.88	9.20	21.84	24.85	6	2240	4.75	4.98	15.43	16.08
1000	51000	10017	4314	6	4314	–	–	2.59	2.96	6	2254	–	–	4.37	4.51
1000	51000	25080	6523	6	6523	2.78	3.58	8.05	9.30	6	2875	1.95	2.12	9.43	9.83
1000	51000	50016	8287	6	8287	6.75	8.79	19.10	22.53	6	3395	4.19	4.31	18.50	19.47
1000	51000	99959	10322	6	10322	16.58	16.97	45.75	53.62	6	3892	6.22	6.35	37.40	39.95
1000	51000	10012	8242	6	8242	–	–	5.03	5.91	6	3769	–	–	8.54	9.01
1000	101000	25065	12830	5	12557	5.67	7.49	17.33	19.72	6	5150	2.37	2.52	19.61	20.13
1000	101000	50051	13699	1	13699	11.98	11.98	37.79	37.79	1	5993	5.24	5.24	35.96	35.96
2500	27500	24957	3790	6	3778	2.22	3.00	2.61	2.88	6	2535	2.05	2.78	3.86	3.91
2500	27500	62446	4587	6	4580	7.22	9.77	6.08	6.87	6	2571	5.99	7.93	6.28	6.37
2500	27500	124902	5181	6	5179	13.99	19.12	12.70	14.93	6	2645	12.57	18.44	10.10	10.25
2500	27500	249558	5946	6	5943	32.79	39.87	26.57	28.57	6	2741	25.37	35.90	17.81	18.01
2500	65000	25060	6120	6	6098	3.36	4.30	3.57	4.23	6	3564	3.11	3.79	6.19	6.42
2500	65000	62503	8590	6	8570	12.44	13.93	10.36	11.49	6	4409	8.44	10.31	12.11	12.53
2500	65000	125196	10546	6	10498	25.70	30.14	24.36	27.86	6	5056	18.75	25.04	21.54	21.96
2500	65000	249806	12610	0	–	–	–	–	–	6	5527	36.30	45.46	39.87	40.50

Pour effectuer la comparaison entre les deux algorithmes, les temps d'exécution et de gestion de la mémoire seront utilisés. À titre d'information, le nombre d'étiquettes traitées par REACH, i.e. celles dont les successeurs sont calculés, et le nombre d'étiquettes remplacées par PULL1 sont indiqués dans les tableaux.

La comparaison des temps d'exécution et de gestion de mémoire pour les problèmes du tableau 1 permet de déterminer si l'algorithme primal-dual PULL1 est encore compétitif face à l'algorithme primal AMPG pour la résolution de problèmes. L'examen des résutlats du tableau 1 montre que les temps d'exécution des deux algorithmes sont en moyenne les mêmes avec un très léger avantage d'environ 5% en moyenne pour PULL1. L'algorithme de pulling est donc légèrement plus rapide que l'algorithme de reaching pour des problèmes avec une contrainte supplémentaire (L=1).

6. RÉSULTATS NUMÉRIQUES DU PULLING MULTIDIMENSIONNEL

L'algorithme de pulling multidimensionnel a été implanté en FORTRAN 77 selon les deux implantations de la section 5. PULL1 utilise des arbres cousus auto-ajustants pour conserver les étiquettes efficaces et utilise une table pour vérifier la relation de dominance entre les étiquettes. PULL2 utilise aussi des arbres cousus auto-ajustants pour conserver les étiquettes efficaces, mais utilise un algorithme spécialisé pour vérifier la dominance. Les deux implantations partagent les mêmes sous-routines, sauf celles vérifiant la dominance. Les deux versions ont été compilées sur le compitaleur FTN5 avec l'option OPT=2 sur les ordinateurs CDC CYBER 173 de l'Université de Montréal. La version PULL1 ne peut être utilisée que si le nombre d'états à chaque noeud, i.e. la taille de la table utilisée pour vérifier la dominance, n'est pas trop grand. La comparaison entre les deux versions ne pourra donc être effectuée que sur de tels problèmes. Il n'existe pas d'implantation de l'algorithme de reaching multidimensionnel.

Toutefois, la comparaison effectuée à la section précédente entre le pulling et le reaching pour le cas d'une seule contrainte supplémentaire montrait que le pulling était légèrement plus rapide que le reaching. De plus, nous avons vu que la complexité de l'implantation du reaching est supérieure à la complexité des implantations du pulling.

La comparaison entre les deux implantations se fera sur des problèmes de type fabrication de journées de travail de conducteurs d'autobus urbains. Ce type de problème doit être résolu lors de la fabrication d'horaires de travail par une méthode de génération de colonnes [DESR84]. Nous voulons produire une journée de travail débutant entre les temps A et B, et respectant trois ou cinq contraintes supplémentaires. Une journée de travail comporte de une à plusieurs tâches séparées par des pauses et doit respecter les contraintes suivantes :

1. le nombre de tâches est limité;
2. la durée totale de la journée (i.e. durée combinée des tâches exécutées et les pauses) est limitée;
3. le nombre d'heures de travail (i.e. durée totale des tâches exécutées) est limité.

De plus, pour les problèmes avec cinq contraintes supplémentaires, deux autres ressources sont contraintes.

Pour chacune de ces contraintes, la quantité de la ressource déjà utilisée pour atteindre une tâche doit appartenir à la fenêtre de réalisabilité associée à la tâche et à cette ressource. Dans notre type de problèmes, dans deux cas (nombre de tâches exécutées, nombre d'heures travaillées) la fenêtre est de la forme [0..bsup]. Dans les trois autres cas (durée totale et les ressources 4 et 5), la fenêtre est adaptée à chaque tâche. La fenêtre sur la durée totale pour une tâche débutant au moment d est [d-b..d-a].

L'ordre des éléments à l'intérieur des vecteurs $(d^1_{ij},\ldots,d^L_{ij})$ est important pour déterminer l'ordre lexicographique et cet ordre détermine l'efficacité de l'algorithme. Nous avons donc choisi, comme premier élément du vecteur, l'élément ℓ tel que $\theta^\ell = \left\{ \dfrac{b^\ell_j - a^\ell_j + 1}{\min\limits_{(i,j)\in A} d^\ell_{ij}} \right\}$ est minimum. De façon à minimiser le nombre maximum de pulling à chaque noeud, θ^ℓ représente le nombre maximum de seaux généralisés si d^ℓ_{ij} est le premier élément du vecteur.

Nous avons produit deux séries de cinq problèmes pour les quatre tailles de problèmes N = 100, 250, 500, 1000. Les caractéristiques de ces problèmes $[a^\ell_i, b^\ell_i]$, $i\in N$, $\ell=1,\ldots,5$ et c_{ij}, d^ℓ_{ij}, $\ell=1,\ldots,5$, $(i,j)\in A$, ont été distribuées selon des lois uniformes pour simuler des problèmes de fabrication de journées de façon similaire aux problèmes de la section précédente. La première série représente des problèmes où les données ont été arrondies pour diminuer la quantité de mémoire v requise à chaque noeud, tandis que dans la seconde série, les données sont plus précises. Ceci permet aussi d'étudier l'effet de v sur le comportement des deux implantations. Nous avons doublé les séries en extrayant aussi des problèmes avec L=3.

La table 2 contient les descriptions des problèmes générés. |N| est le nombre de noeuds dans chaque problème. |A| est le nombre moyen d'arcs réalisables dans le réseau. Il est à noter que le nombre d'arcs réalisables est plus grand dans les problèmes avec trois contraintes que dans ceux avec cinq contraintes. Ceci est dû aux vérifications de réalisabilité (6) additionnelles. v est le nombre d'états à un noeud alors que V est le nombre total d'états. V est le nombre maximum d'étiquettes possibles. |E| est le nombre d'étiquettes associées à des chemins efficaces dans la solution finale. Il est fréquent que le nombre d'étiquettes présentes dans la solution lors de l'exécution de l'algorithme soit supérieur à |E|.

Les statistiques d'exécution présentées dans la table 2 sont le nombre de pulling et les temps d'exécution.

Les statistiques d'exécution des 16 séries de problèmes sont contenues dans la table 2. La première constatation est que PULL2 a pu résoudre 15 des 16 séries de problèmes, alors que PULL 1 n'a pu résoudre, faute de mémoire centrale suffisante, que 11 des 16 séries. La série qu'aucune des implantations n'a pu résoudre demandait beaucoup plus de mémoire centrale que les autres. Les deux implantations semblent être complémentaires. PULL1 est légèrement meilleur que PULL2 sur les problèmes avec L=3 et ou v est petit, cet avantage disparaît lorsque v grandit. Dans le cas ou L=5, la domination est claire, PULL2 demande de 5% à 20% du temps de calcul de PULL1 et moins de mémoire centrale.

Il est possible d'expliquer la différence de temps d'exécution de PULL1 et PULL2 par la nature des variables présente dans l'analyse en pire cas de leur complexité. Pour PULL1, cette complexité est d'ordre $O(VL(n\log v+v))$, et d'ordre $O(VL(n\log v+\log^2 v))$ pour PULL2. La partie $O(VLn\log v)$ est identique pour les deux implantations, seule la partie $O(VLv)$ de PULL1 diffère de la partie $(VL\log^2 v)$ de PULL2. La vérification de la dominance dans PULL1 demande au moins $O(v)$ étapes alors que dans le cas de PULL2, elle demande $O(k\log^2 k)$ où k est le nombre de vecteurs obtenus ($k< v$). Dans le cas où v est grand et k est petit, $O(v)$ est souvent plus grand que $O(k\log^2 k)$, ce qui explique que PULL2 soit plus rapide que PULL1 dans le cas où L=5.

Le comportement de l'algorithme PULL2 est surprenant à un autre point de vue : son temps d'exécution diminue lorsque l'on augmente le nombre de contraintes de 3 à 5 et que le nombre d'états V a considérablement augmenté. Cette diminution s'explique par la réduction du nombre d'arcs

TABLEAU 2

Statistiques d'exécution des deux implantations

DATA DESCRIPTION					PULL1		PULL2	
|N|	|A|	V	|E|	#PULL	Temps UC (sec.)		Temps UC (sec.)	
					Max	Moy	Max	Moy
Problèmes de trois contraintes avec v variant entre 1 et 200 (moy =92)								
100	1566	9224	164	101	1,551	1,214	1,011	0,890
250	9096	23086	526	251	6,029	4,612	5,014	4,294
500	36241	46210	1290	501	21,717	16,055	23,071	17,411
1000	144082	92336	2998	1001	86,674	59,645	111,348	69,747
Problèmes de trois contraintes avec v variant entre 1 et 1000 (moy = 427)								
100	1746	42724	348	101	4,953	3,235	1,253	1,097
250	10282	107166	1657	251	17,890	10,509	13,732	6,973
500	40836	214416	4578	501	60,657	34,586	77,735	31,816
1000	espace mémoire insuffisant							
Problèmes de cinq contraintes avec v variant entre 1 et 9200 (moy = 1298)								
100	582	133833	83	101	18,756	10,522	0,602	0,456
250	3364	325658	401	251	47,211	27,211	2,479	2,062
500	12030	646179	1154	501	97,826	55,956	9,677	7,050
1000	46889	1295252	2913	1001	205,787	121,025	39,681	26,371
Problèmes de cinq contraintes avec v variant entre 1 et 46000 (moy = 6012)								
100	641	602812	123	101	espace mémoire insuffisant		0,686	0,517
250	3742	1511336	729	251			3,744	2,592
500	13423	2998202	2513	501			22,712	10,959
1000	52978	6009189	7641	1001			104,709	46,806

réalisables dans les problèmes entraînant une réduction du nombre d'états examinés. Le nombre d'opérations de PULL2 dépend du nombre d'étiquettes examinées plutôt que du nombre d'états. PULL2 permet la résolution de problèmes fortement contraints en un temps raisonnable.

CONCLUSION

L'algorithme de reaching est une généralisatin évidente des algorithmes de plus court chemin. Dans cet algorithme, il y a un nombre égal des deux opérations de prolongement d'un chemin : 1^0 calcul du coût d'un successeur et 2^0 mise à jour du successeur. Dans le cas des problèmes de plus court chemin, il y a un état par noeud et ces deux opérations s'effectuent en temps constant. Toutefois, dans les problèmes de programmation dynamique où il y a plus d'un état par noeud, le nombre d'opérations nécessaires pour mettre à jour les états du successeur dépend du nombre d'états par noeud : v.

L'algorithme de pulling a été mis au point pour obtenir un meilleur équilibre entre le nombre de chacune de ces deux opétations. L'algorithme de pulling cherche à minimiser le nombre de mises à jour des états du successeur. L'algorithme de pulling a donc une meilleure complexité que l'algorithme de reaching et les résultats numériques montrent que, dans le cas d'une seule contrainte supplémentaire, le pulling est déjà supérieur au reaching.

RÉFÉRENCES

[ANEJ 83] ANEJA, Y.P., AGGARWAL, V. et NAIR, K.P.K., Shortest chain subject to side constraints. Networks 13 (1983) 295-302

[DESR 86] DESROCHERS, M., La fabrication d'horaires de travail pour les conducteurs d'autobus par une méthode de génération de colonnes. Thèse de doctorat, département d'informatique et de recheche opérationnelle, Université de Montréal, dépôt prévu : mai 1986.

[DESR 83] DESROSIERS, J., PELLETIER, P. et SOUMIS, F., Plus court chemin avec contraintes d'horaires. R.A.I.R.O. Recherche Opérationnelle 17 (1983) 357-377.

[DESR 84] DESROSIERS, J., SOUMIS, F. et DESROCHERS, M., Routing with time windows by column generation. Networks 14 (1984) 545-565.

[HABE 81] HABENICHT, W., Efficient routes in vector-values graphs. Proceedings of the Seventh Conference on Graphtheoretic Concepts in Computer Science, J.R. Mühlbacher, ed., Carl Hanser Verlag, Munich (1981) 349-355.

[HANS 80] HANSEN, P., Bicriterion path problems, dans G. Fandel et T. Gal, éds., Multiple Criteria Decision Making : Theory and Applications, Lecture Notes in Economics and Mathematical Systems 177, Springer Verlag, Heidelberg (1980) 109-127.

[JAFF 84] JAFFE, J.M., Algorithms for finding paths with multiple constraints. Networks 14 (1984) 95-116.

[JOKS 66] JOHSCH, H.C., The shortest route problem with constraints. Journal of Mathematical Analysis and Applications 14 (1966) 191-197.

[KUNG 75] KUNG, H.T., LUCCIO, F., PREPARATA, F.P., On finding the maxima of a set of vectors, JACM 22, 5 (1975) 469-476.

[MART 84] MARTINS, E.Q.V., On a multicriteria shortest path problem. European Journal of Operational Research 16 (1984) 236-245.

[MINO 75] MINOUX, M., Plus court chemin avec contraintes : algorithmes et applications. Annales des Télécommunications 30, 12 (1975) 1-12.

[SLEA 83] SLEATOR, D.D., TARJAN, R.E., Self-adjusting binary trees, Proc. Fifteenth Annual ACM Symposium on the Theory of Computing (1983) 235-245.

[VINC 74] VINCKE, P., Problèmes multicritères. Cahiers du Centre d'Études en Recherche Opérationnelle 16 (1974) 425-439.